Symbolic Semantics for CSP

Liyi Li, Elsa Gunter, and William Mansky
Department of
Computer Science, University of
Illinois at Urbana-Champaign
{liyili,egunter,mansky1}@illinois.edu

Communicating Sequential Processes (CSP) is a well-known formal language for describing concurrent systems. Brookes, Hoare and Roscoe [2] have given a transition semantics for CSP that underlies common approaches to model checking properties of CSP programs. In this paper, we present a generalized transition semantics of CSP, which we call HCSP, that merges the original transition system with ideas from Floyd-Hoare Logic and symbolic computation. This generalized semantics is shown to be sound and complete with respect to the original trace semantics. Traces in our system are symbolic representations of families of traces as given by the original semantics. This more compact representation allows us to expand the original CSP systems to effectively model check some CSP programs which are difficult for other CSP systems to analyze. In particular, our system can handle certain classes of non-deterministic choices as a single transition, while the original semantics would treat each choice separately, possibly leading to large or unbounded case analyses. All work described in this paper has been carried out in the theorem prover Isabelle. This then provides us with a framework for automated and interactive analysis of CSP processes. It also give us the ability to extract Ocaml code for an HCSP-based simulator directly from Isabelle.

1 Motivation

Communicating Sequential Processes (CSP) is a kind of process algebra for modeling concurrency. When using CSP to model a complex system, such as the medical mediator system in Gunter *et al.* [3], a common problem is that one piece of a CSP process can generate a large number of similar actions. For example, the CSP process $c?x : B \to P$ is generally modeled as receiving a single value from the set $\{x|B\}$. If B describes a nonempty set, the execution of the process will depend on the size of the set in traditional CSP tools. In practice, if the set $\{x|B\}$ is an infinite set, many CSP simulators will actually create an endless number of similar processes and wait for other parts of the program to stop these processes. In this paper, we present Holistic CSP (HCSP), a new semantics for CSP processes that uses a symbolic representation of actions to capture a group of properties simultaneously instead of considering only a single element with a single property.

The approach we take in this work is to represent families of transitions in CSP by a single transition in HCSP. In some implementations of CSP simulators, such as the JCSP package in Java, actions are implemented as objects, allowing sets and other complex objects to be passed as actions through channels. However, this is still a single action instead of a property describing a group of actions and corresponding transitions. In HCSP, we can view a set of actions as a whole in some contexts, but also divide it based on various properties in other contexts.

The differences between the original CSP semantics and the HCSP semantics are large enough to be visible in some very simple examples. Four such examples are shown in Figure 1. We tested these examples by using FDR2 to determine whether the processes refine themselves. FDR2 takes several seconds to run process A, and several minutes to finish the proof for process B. It fails to terminate on

 © Li, Gunter, and Mansky

processes C and D. On process C it begins to run, but eventually faces a stack overflow; it claims that process D exceeds the integer limit for FDR2, even though the FDR2 User Manual suggests that 32-bit integers can be handled [8]. Also, when we directly input process C into the CSPM-based simulator ProBE, the whole program crashed without leaving any information. However, the HCSP simulator we have derived from the semantics given in this paper takes two or three seconds to list all the traces for all four processes. From these examples, we can clearly see that the running time of FDR2 depends on the size of the set in each process. These facts reflect, in part, that the original CSP and Machine-Readable

$$A.\quad (\prod_{x:x>0\wedge x<10} \mathsf{SKIP}) [\![\{k.x|x>0\wedge x<100\}]\!] (\prod_{x:x>0\wedge x<50} \mathsf{SKIP}) \sqcap (\prod_{x:x>50\wedge x<100} \mathsf{SKIP})$$

$$B.\quad (\prod_{x:x>0\wedge x<10*10^3} \mathsf{SKIP}) [\![\{k.x|x>0\wedge x<100*10^3\}]\!] (\prod_{x:x>0\wedge x<50*10^3} \mathsf{SKIP}) \sqcap (\prod_{x:x>50*10^3\wedge x<100*10^3} \mathsf{SKIP})$$

$$C.\quad (\prod_{x:x>0\wedge x<1000} \mathsf{SKIP}) [\![\{k.x|x>0\wedge x<100\}]\!] (\prod_{x:x>0} \mathsf{SKIP}) \sqcap (\prod_{x:x\geq 50} \mathsf{SKIP})$$

$$D.\quad (\prod_{x:x>0\wedge x<10*10^6} \mathsf{SKIP}) [\![\{k.x|x>0\wedge x<100*10^6\}]\!] (\prod_{x:x>0\wedge x<50*10^6} \mathsf{SKIP}) \sqcap (\prod_{x:x>50*10^6\wedge x<100*10^6} \mathsf{SKIP})$$

Figure 1: Example

CSP (CSPM) semantics do not have rules for replicated operators (Replicated Internal Choice, Replicated External Choice, etc). Instead, they view replicated operators as macros of their binary versions over sets. This fact means that the original CSP and CSPM syntax cannot express a replicated operator if the set of the replicated operator is infinite, such as the second and third Replicated Internal Choice operators in process C. Even if the set is finite, the cost is very expensive for CSP-semantics-based tools to run a small replicated process in a large macro, such as the process D. On the other hand, HCSP-semantics-based tools can overcome this problem and run CSP processes regardless of the size of sets in replicated operators. By using HCSP semantics, the four processes in the example will have the same number of possible next moves. In addition, HCSP-semantics-based tools can expand the set of possible CSP processes to execute, processes C and D above are examples of this fact. We will see that this fact is useful in some real applications such as the medical mediator system in Gunter *et al.* [3].

This paper's contribution is a new formalization of CSP (syntax in Section 2 and semantics in Section 3), which we have proved correct (proof sketched in Section 4) and supports simulation (Section 5). Along with the semantics in Section 3, we will also talk about how to transfer a rule in original CSP into HCSP by adding additional information to the transitions. Before that, we demonstrate the expressiveness and power of HCSP as compared to traditional CSP semantics by using examples. Finally, we reason about where the power comes from in Section 7 and provide one more example in Section 6 to show that there is a large collection of problems, which are hard to be analyzed in CSP framework, can be reasoned in HCSP framework. All of the work described in this paper has been formalized and proved in the interactive theorem prover Isabelle/HOL [9].

2 Syntax

The syntax of HCSP is given in Figure 2. For the remainder of this paper, the following name conventions will be used. We will use P and Q for processes. Lower case p refers to an HCSP process name. The

$P =$

	Ω	Successful termination	SKIP	Awaiting successful termination
$\mid$	STOP	Unexpected termination	$c.a \rightarrow P$	Prefix by action a on channel c
$\mid$	if b then P else Q	If statement	$\$p$	Process name p as prcoess
$\mid$	$P;Q$	Sequential execution	$P \Box Q$	Binary external choice
$\mid$	$P \sqcap Q$	Binary internal choice	$\prod_{x:B} P$	Replicated internal choice
$\mid$	$c?x : B \rightarrow P$	External set prefix	$\square_{x:B} P$	Replicated external choice
$\mid$	$P[\![\{k.x\|B\}]\!]Q$	Parallel composition	$P \setminus \{k.x\|B\}$	Hiding over a set of actions

Figure 2: HCSP Syntax

c represents an HCSP channel, while the a represents an HCSP action. In HCSP in Isabelle, a channel and an action have the same user-defined type, which we will refer to here as the "act" type. The B is a proposition describing the property of a set. In HCSP, we include both *variables* and *parameters*, which are distinct types. We use k for variables ranging over HCSP channels and x for variables over actions. We use U and V to refer to parameters ranging over channels and actions. Variables and parameters serve similar functions, but differ as follows: variables may occur free or bound in HCSP processes and may be replaced by actions or channels by substitution, while parameters occur essentially as local constants not subject to binding or substitution. In the rest of the paper, we will use *freeParams* to refer to a function returning all free parameters in an expression of arbitrary type. Transitions will be labeled by $\surd$, τ (see Section 3) or pairs of parameters, one for a channel and one for an action. We will use l to represent a transition label. Finally, the Greek letter ρ refers to an assignment function that assigns values to parameters.

One remark that must be made here concerns the scope of variables. In the processes $c?x : B \rightarrow P$, $\prod_{x:B} P$ and $\square_{x:B} P$, the scope of variable x is both the proposition B and the process P, while the scope of the variables k and x is only the proposition B in the processes $P[\![\{k.x|B\}]\!]Q$ and $P \setminus \{k.x|B\}$.

The syntax of HCSP differs from that of CSPM by Bryan Scattergood [11] in three ways. Firstly, the actions of CSP are explicitly divided into channels and actions (written $c.a$) in HCSP syntax. Secondly, for the sets used in constructs such as the parallel composition of two processes or replicated internal choice, we use a set comprehension notation. This decomposition of sets into variables and predicates will facilitate the statement of the transition rules of HCSP semantics. Finally, HCSP currently lacks the CSP Renaming operator.

HCSP is parameterized by four user-defined types: a type of expressions for actions and channels (acts), a type of propositions, a type of process names and a type of values to be assigned to acts and boolean expressions. The language of boolean expressions must minimally support conjunction, negation, equality, and a special operator Wf ranging over boolean expressions, while the language of acts must minimally support a function injecting the disjoint sum of variables and parameters into acts. The Wf operator checks whether a given proposition is well-formed in a given environment, in the sense that it can be evaluated to a boolean value in that environment. In the semantics section, we will see how Wf is useful in distinguishing between the different interpretations of boolean expressions in the If-then-else operator versus in the Parallel operator in HCSP.

In some cases, the advantages of HCSP will cause significant differences when we are trying to

$$\begin{aligned}
\mathsf{System} = \quad & ((\|\!|_{m} (\sqcap_{n}(\mathsf{Nurse}(n,m) \; |[\{\{\mathsf{HCI}^n_m.x|\mathsf{True}\}\}]| \\
& \mathsf{Med}(n,m)) \; |[\{\{\mathsf{EHRCh}_m.x|\mathsf{True}\}\}]| \\
& \mathsf{EHRInterface})) \setminus \{y|\forall n\ m\ p\ d\ x. \\
& y \neq \mathsf{RFIDChan}^n_p.x \wedge y \neq \mathsf{EHRBECh}^m.x \wedge \\
& y \neq \mathsf{BTAddr}^{n,m}_d.x \ \wedge y \neq \mathsf{BTScan}_m.x\}); \\
& \mathsf{System}
\end{aligned}$$

$$\begin{aligned}
\mathsf{Med}(n,m) = \mathsf{HCI}^n_m?\mathsf{GetID} \rightarrow \mathsf{HCI}^n_m?x_1 \rightarrow x_1?y_1 \rightarrow \\
\mathsf{HCI}^n_m?x_2 \rightarrow x_2?y_2 \rightarrow \mathsf{EHRCh}^m!(y_1,y_2) \rightarrow \mathsf{EHRCh}^m?n_1 \\
\rightarrow \text{ if } n_1 = \mathsf{Error} \text{ then } \mathsf{HCI}^n_m!n_1 \rightarrow \mathsf{HCI}^n_m?\mathsf{OK} \rightarrow \mathsf{Skip} \\
\text{else } \mathsf{EHRCh}^m?n_2 \rightarrow \mathsf{HCI}^n_m!(n_1,n_2) \rightarrow \mathsf{HCI}^n_m?z \rightarrow \\
\text{if } z = \mathsf{Yes} \text{ then } \mathsf{MedRead}(n,m) \text{ else} \mathsf{Skip}
\end{aligned}$$

$$\begin{aligned}
\mathsf{Nurse}(n,m) = \\
\mathsf{HCI}^n_m!\mathsf{GetID} \rightarrow \sqcap_{p}(\mathsf{HCI}^n_m!(\mathsf{RFIDChan}^{n,m}_p) \rightarrow \\
\sqcap_{d}(\mathsf{HCI}^n_m!(\mathsf{RFIDChan}^{n,m}_d) \rightarrow \mathsf{HCI}^n_m?x \rightarrow \\
\text{if } x = (\mathsf{Name}(p),\mathsf{Name}(d)) \\
\text{then } (\mathsf{HCI}^n_m!\mathsf{Yes} \rightarrow \mathsf{TakeCkReading}(n,m,p,d)) \\
\text{else if } x = \mathsf{Error} \text{ then } (\mathsf{HCI}^n_m!\mathsf{OK} \rightarrow \mathsf{Skip}) \\
\text{else } (\mathsf{HCI}^n_m!\mathsf{No} \rightarrow \mathsf{Skip})))
\end{aligned}$$

$$\mathsf{BTDevs} = \quad \Box_{m}(\sqcap_{X \subseteq \mathsf{BTDevices}} \mathsf{BTScan}_m!X \rightarrow \mathsf{BTDevs})$$

Figure 3: Part of Medical Mediator Project Code [3]

analyze traces through CSP tools. One such example is the medical mediator project [3]. The medical mediator project provides a formal model of the use of a device for Automated Identification and Data Capture (AIDC) for vital signs measurements in hospitals [3]. To demonstrate our HCSP semantics, we provide a small piece of the CSP code for the medical mediator project in Figure 3 and describe here the operators occurred in the code. The Nurse process models all behaviors that nurses can do with the mediator, while the Med and the unlisted MedRead processes model the identification system. The unlisted EHRInterface process models the backend system for delivering information displayed on the mediator and collecting data transmitted from the mediator. The unlisted TakeCKReading process is used in the Nurse process to model the behavior by which a nurse verifies a reading from a patient [3]. The BTDevs process models the behavior of announcing Bluetooth channels to the mediators scanning for them. It is part of the unlisted process Given. Finally, the System process combines the Nurse process, the Med process, and the EHRInterface process together using the Parallel operator, and limits the set of actions that these three processes can communicate to the set in the middle of the Parallel operator.

The Replicated Interleaving operator, $\|\!|_{m} P$, in the System process uses the CSP macro $\|\!|_{x:m} P = P[m_1/x] \; |[\{\}]| P[m_2/x] |[\{\}]| ...$, where $m_1, m_2, ... \in \{x|m\}$. It means that all its sub-processes do work individually and do not communicate at all. We keep the same macro in HCSP semantics because it would allow the infinite parallel processes occurred in a program if we allowed infinite set in the Replicated Interleaving operator, which is a fundamental change for the meaning of the CSP language. In the original CSP definition, the External Set Prefix operator, $c?x : B \rightarrow P$, is defined as a macro as $\Box_{x:B} c.x \rightarrow P$. In HCSP, we provide the operator with a semantic interpretation because the operator is useful in representing "receiving" in CSP and it generates fewer vacuous rule applications than the specialized instance of the Replicated External Choice operator. Also, the $c!a \rightarrow P$ operator that appears in the Nurse process of the medical mediator project is the same as $c.a \rightarrow P$, as according to the CSP book [5].

In the CSP world, there is no notion of sending and receiving messages; instead, they use the notion of internal choice and external choice to represent sending and receiving. For example, in the Nurse process, the operator $\mathsf{HCI}^n_m!\mathsf{RFIDChan}^{n,m}_p \rightarrow P$ means that we internally commit an action $\mathsf{RFIDChan}^{n,m}_p$ on the channel HCI^n_m, where both action and channel are previously internally selected. select a value GetID for the HCI^n_m channel. On the other hand, in the Med process, the operator $\mathsf{HCI}^n_m?x_1 \rightarrow \mathsf{P}$ means

that we externally wait to receive every possible value through the HCI_m^n channel. The the operator $\mathsf{HCI}_m^n?x_1 \to \mathsf{P}$ uses an abbreviation of x_1 as $x_1 : \mathsf{True}$, where we omit the set predicate True here. In addition, in the System process, we select a mediator from the Replicated Interleaving operator and a nurse from the Replicated Internal Choice, then select the corresponding channel from the HCI channels; this channel will be used in the Nurse and Med processes as the HCI_m^n channel. Then, these two processes, Nurse and Med, can communicate via the Parallel operator in the System process because the set of the Parallel operator includes all communication along all the HCI_m^n channel. The Parallel operator $\mathsf{Nurse}(n,m)\|[\{\mathsf{HCI}_m^n.x|True\}]\|\mathsf{Med}(n,m)$ allows interleaving actions to happen if the actions from $\mathsf{Nurse}(n,m)$ and $\mathsf{Med}(n,m)$ do not belong to the middle set, while it allows communication between processes $\mathsf{Nurse}(n,m)$ and $\mathsf{Med}(n,m)$ through the same action if the action belongs to the middle set. Finally, the Hiding operator is used to make a transition become local, i.e., invisible to the outside world; it provides security for the process involved. For example, the System process hides a transition if the transition's belongs to either of the HCI or EHRCh channels. This means that the System process does not want outside processes to see and affect the actions along the HCI and EHRCh channels.

3 Semantics

$$(\alpha,\gamma,\Omega\|[\{k.x|B\}]\|\Omega) \xrightarrow{\surd} (\alpha',\gamma',\Omega) \quad \text{Par_omega}$$

$$(\alpha,\gamma,\mathsf{SKIP}) \xrightarrow{\surd} (\alpha,\gamma,\Omega) \quad \text{Skip} \qquad (\alpha,\gamma,\$p) \xrightarrow{\tau} (\alpha,\gamma,procEnv(p)) \quad \text{Proc_name}$$

$$(\alpha,\gamma,P\sqcap Q) \xrightarrow{\tau} (\alpha,\gamma,P) \quad \text{Int_choice1} \qquad (\alpha,\gamma,P\sqcap Q) \xrightarrow{\tau} (\alpha,\gamma,Q) \quad \text{Int_choice2}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{\tau} (\alpha',\gamma',P')}{(\alpha,\gamma,P\Box Q) \xrightarrow{\tau} (\alpha',\gamma',P'\Box Q)}\ \text{Ext_choice_tau1} \qquad \frac{(\alpha,\gamma,Q) \xrightarrow{\tau} (\alpha',\gamma',Q')}{(\alpha,\gamma,P\Box Q) \xrightarrow{\tau} (\alpha',\gamma',P\Box Q')}\ \text{Ext_choice_tau2}$$

$$\frac{l\neq\tau \quad (\alpha,\gamma,P) \xrightarrow{l} (\alpha',\gamma',P')}{(\alpha,\gamma,P\Box Q) \xrightarrow{l} (\alpha',\gamma',P')}\ \text{Ext_choice1} \qquad \frac{l\neq\tau \quad (\alpha,\gamma,Q) \xrightarrow{l} (\alpha',\gamma',Q')}{(\alpha,\gamma,P\Box Q) \xrightarrow{l} (\alpha',\gamma',Q')}\ \text{Ext_choice2}$$

$$\frac{l\neq\surd \quad (\alpha,\gamma,P) \xrightarrow{l} (\alpha',\gamma',P')}{(\alpha,\gamma,P;Q) \xrightarrow{l} (\alpha',\gamma',P';Q)}\ \text{Seq_step} \qquad \frac{(\alpha,\gamma,P) \xrightarrow{\surd} (\alpha',\gamma',P')}{(\alpha,\gamma,P;Q) \xrightarrow{\tau} (\alpha',\gamma',Q)}\ \text{Seq_check}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{\tau} (\alpha',\gamma',P')}{(\alpha,\gamma,P\setminus\{k.x|B\}) \xrightarrow{\tau} (\alpha',\gamma',P'\setminus\{k.x|B\})}\ \text{Hid_tau} \qquad \frac{(\alpha,\gamma,P) \xrightarrow{\surd} (\alpha',\gamma',P')}{(\alpha,\gamma,P\setminus\{k.x|B\}) \xrightarrow{\surd} (\alpha',\gamma',\Omega)}\ \text{Hid_omega}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{\surd} (\alpha',\gamma',P')}{(\alpha,\gamma,P|[\{k.x|B\}]\|Q) \xrightarrow{\tau} (\alpha',\gamma',\Omega|[\{k.x|B\}]\|Q)}\ \text{Par_check1} \qquad \frac{(\alpha,\gamma,Q) \xrightarrow{\surd} (\alpha',\gamma',Q')}{(\alpha,\gamma,P|[\{k.x|B\}]\|Q) \xrightarrow{\tau} (\alpha',\gamma',P|[\{k.x|B\}]\|\Omega)}\ \text{Par_check2}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{\tau} (\alpha',\gamma',P')}{(\alpha,\gamma,P|[\{k.x|B\}]\|Q) \xrightarrow{\tau} (\alpha',\gamma',P'|[\{k.x|B\}]\|Q)}\ \text{Par_tau1} \qquad \frac{(\alpha,\gamma,Q) \xrightarrow{\tau} (\alpha',\gamma',Q')}{(\alpha,\gamma,P|[\{k.x|B\}]\|Q) \xrightarrow{\tau} (\alpha',\gamma',P|[\{k.x|B\}]\|Q')}\ \text{Par_tau2}$$

Figure 4: HCSP semantics (category 1st)

In order to describe the HCSP semantics, there are some functions that need to be supplied for the evaluations of user-defined types. A function *procEnv* must be defined for interpreting a given process name p, where the process name p in the Proc_name operator (denoted by \$ in the concrete syntax in

$$\frac{\exists \rho \,.\, \rho \models (b \wedge \gamma)}{(\alpha,\gamma,\text{if } b \text{ then } P \text{ else } Q) \xrightarrow{\tau} (\alpha, b \wedge \gamma, P)}\ \text{Ifthenelse1} \qquad \frac{\exists \rho \,.\, \rho \models (\neg b \wedge \mathsf{Wf(b)} \wedge \gamma)}{(\alpha,\gamma,\text{if } b \text{ then } P \text{ else } Q) \xrightarrow{\tau} (\alpha, \neg b \wedge \mathsf{Wf(b)} \wedge \gamma, Q)}\ \text{Ifthenelse2}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{(U.V)} (\alpha',\gamma',P') \quad \exists \rho \,.\, \rho \models (\neg B[U/k][V/x] \wedge \gamma')}{(\alpha,\gamma,P \setminus \{k.x|B\}) \xrightarrow{(U.V)} (\alpha', \neg B[U/k][V/x] \wedge \gamma', P' \setminus \{k.x|B\})}\ \text{Hid_neg}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{(U.V)} (\alpha',\gamma',P') \quad \exists \rho \,.\, \rho \models (B[U/k][V/x] \wedge \gamma')}{(\alpha,\gamma,P \setminus \{k.x|B\}) \xrightarrow{\tau} (\alpha', B[U/k][V/x] \wedge \gamma', P' \setminus \{k.x|B\})}\ \text{Hid_pos}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{(U.V)} (\alpha',\gamma',P') \quad \exists \rho \,.\, \rho \models (\neg B[U/k][V/x] \wedge \gamma')}{(\alpha,\gamma,P[\![\{k.x|B\}]\!]Q) \xrightarrow{(U.V)} (\alpha', \neg B[U/k][V/x] \wedge \gamma', P'[\![\{k.x|B\}]\!]Q)}\ \text{Par_out1}$$

$$\frac{(\alpha,\gamma,Q) \xrightarrow{(U.V)} (\alpha',\gamma',Q') \quad \exists \rho \,.\, \rho \models (\neg B[U/k][V/x] \wedge \gamma')}{(\alpha,\gamma,P[\![\{k.x|B\}]\!]Q) \xrightarrow{(U.V)} (\alpha', \neg B[U/k][V/x] \wedge \gamma', P[\![\{k.x|B\}]\!]Q')}\ \text{Par_out2}$$

$$\frac{(\alpha,\gamma,P) \xrightarrow{(U.V)} (\alpha',\gamma',P') \quad (\alpha',\gamma',Q) \xrightarrow{(U'.V')} (\alpha'',\gamma'',Q') \quad \exists \rho \,.\, \rho \models (U = U' \wedge V = V' \wedge B[U/k][V/x] \wedge \gamma'')}{(\alpha,\gamma,P[\![\{k.x|B\}]\!]Q) \xrightarrow{(U.V)} (\alpha'', U = U' \wedge V = V' \wedge B[U/k][V/x] \wedge \gamma'', P'[\![\{k.x|B\}]\!]Q')}\ \text{Par_in}$$

Figure 5: HCSP semantics (categories 2nd)

Figure 2) is user-defined, and can be arbitrarily complicated. A family of substitution functions $T[a/x]$ is needed for the replacement of variables by acts in each of acts, boolean expressions, and process names. Using these, we define the substitution function for processes. There also needs to be a family of user-defined evaluation functions for acts and boolean expressions and a "models" function, $\models$, for checking whether a boolean expression is true under a given assignment function.

The semantics for HCSP is given in Figures 4 to 6. It is a merge of the original CSP semantics given by Roscoe *et al.* [10] with ideas from Floyd-Hoare Logic [4] and symbolic computation. We present a labeled transition system for HCSP over triples of the form (α,γ,P), where P is an HCSP process, γ is a proposition in HCSP that is intended to state the current requirements for parameters "in scope", including those occurring free in P, and α is a set of parameters large enough to contain all parameters occurring free in P or γ. The proposition γ plays the role of providing the pre- and post-condition for each transition. We carry α with us to allow for the choice of fresh parameter names guaranteed not to clash with a potentially bigger scope than the one locally presented by P and γ.

The main contribution of the paper is to provide a basic framework to translate transition semantics into symbolic semantics by merging state information in Floyd-Hoare logic [4] with the existing semantics. The basis of our approach is to associate a state environment whose structure is implemented as a predicate with an evaluation of the semantics. When we are going to evaluate a statement in a programming language, instead of evaluating the statement directly, we view each transition as a constraint which can be merged into the current environment. After updating the environment, we can reason about the evaluation by asking about the satisfiability of the environment predicate. For example, if we translate the traditional semantics of CSP by our framework, then the new semantics tells the users the range of values for a particular next possible move in CSP when the environment predicate is satisfiable, or a next move is impossible because the environment predicate associated with it is unsatisfiable. A trace in

$$\frac{U \notin \alpha \quad V \notin \{U\} \cup \alpha \quad \exists \rho \,.\, \rho \models (U = c \wedge V = a \wedge \gamma)}{(\alpha, \gamma, c.a \to P) \xrightarrow{(U.V)} (\{U,V\} \cup \alpha, U = c \wedge V = a \wedge \gamma, P)}\ \textsf{Act_prefix}$$

$$\frac{U \notin \alpha \quad V \notin \{U\} \cup \alpha \quad \exists \rho \,.\, \rho \models (U = c \wedge B[V/x] \wedge \gamma)}{(\alpha, \gamma, c?x : B \to P) \xrightarrow{(U.V)} (\{U,V\} \cup \alpha, U = c \wedge B[V/x] \wedge \gamma, P[V/x])}\ \textsf{Ext_prefix}$$

$$\frac{U \notin \alpha \quad \exists \rho \,.\, \rho \models (B[U/x] \wedge \gamma)}{(\alpha, \gamma, \prod_{x:B} P) \xrightarrow{\tau} (\{u\} \cup \alpha, B[U/x] \wedge \gamma, P[U/x])}\ \textsf{Rep_int_choice}$$

$$\frac{U \notin \alpha \quad (\{u\} \cup \alpha, B[U/x] \wedge \gamma, P[U/x]) \xrightarrow{\tau} (\alpha', \gamma', P') \quad \exists \rho \,.\, \rho \models \gamma'}{(\alpha, \gamma, \Box_{x:B} P) \xrightarrow{\tau} (\alpha', \gamma', (\Box_{x:B \wedge x \neq U} P) \Box P')}\ \textsf{Rep_ext_tau}$$

$$\frac{U \notin \alpha \quad l \neq \tau \quad (\{u\} \cup \alpha, B[U/x] \wedge \gamma, P[U/x]) \xrightarrow{l} (\alpha', \gamma', P') \quad \exists \rho \,.\, \rho \models \gamma'}{(\alpha, \gamma, \Box_{x:B} P) \xrightarrow{l} (\alpha', \gamma', P')}\ \textsf{Rep_ext_nor}$$

Figure 6: HCSP semantics (category 3rd)

this semantics is an execution pattern corresponding to a possibly large or even infinite set of individual execution traces in the original CSP semantics. For instance, the HCSP semantics allows a potentially infinite collection of data in the set of replicated choice operators to execute as one single transition, which is a nice feature that traditional CSP-based analysis tools cannot provide.

The labels of the HCSP semantics will be ranged over by l as follows:

$$l = \surd \mid \tau \mid (U.V)$$

The label $\surd$ represents process completion, the label τ represents a process performing an invisible action, and the label $(U.V)$ represents a pair of a channel and a real action. In any execution of a process in accordance with this semantics, the sequence of transitions is labeled with mutually distinct pairs of parameters $(U.V)$, when not labeled by $\surd$ or τ. The values potentially represented by labels of the form $(U.V)$ are progressively restricted by the conditions in each of the subsequent triples resulting from each transition in the execution. In this way, a single execution in the transition semantics of HCSP potentially represents a parameterized family of executions from the original CSP semantics.

When translating the original CSP semantics into HCSP semantics, the main task is to merge information about actions and channels into the environment condition γ, and use this condition when we are trying to evaluate a CSP process. To do so, we will divide the transition rules in CSP into three categories: rules for basic operators having no side conditions other than $\surd$-τ label constraints, rules with side conditions needing to be translated into the HCSP framework, and rules for operators that were treated as macros in CSP.

The rules without side conditions are those for SKIP, STOP, Internal Choice, External Choice and Sequential Composition operators. For these, we just need to add the set of free parameters and environment conditions to the processes involved, propagating constraints from hypotheses to conclusions in the original CSP rules. For example, the External Choice operator has the following semantics in the

original CSP:

$$\frac{l \neq \tau \quad P \xrightarrow{l} P'}{P \Box Q \xrightarrow{l} P'} \quad \text{becomes} \quad \frac{l \neq \tau \quad (\alpha,\gamma,P) \xrightarrow{l} (\alpha',\gamma',P')}{(\alpha,\gamma,P \Box Q) \xrightarrow{l} (\alpha',\gamma',P')}\ \textsf{Ext_choice1}$$

The second category contains the operators with set or boolean guard information, namely, Parallel, Hiding and If-then-else. The general idea for translating the rules of CSP into HCSP is to treat the set information and boolean guards as new restrictions on the environment and merge them into the conditions as post-conditions of the transitions after we do the same steps to translate the rules as were done with the first category. This requires us to translate all set information into set comprehension notation. For example, for the Parallel operator, we make the following translation:

$$\frac{l \in X \quad P \xrightarrow{l} P' \quad Q \xrightarrow{l} Q'}{P|[X]|Q \xrightarrow{l} P'|[X]|Q'} \quad \text{becomes}$$

$$\frac{\begin{array}{c}\exists \rho \,.\, \rho \models (U = U' \wedge V = V' \wedge B[U/k][V/x] \wedge \gamma'') \\ (\alpha,\gamma,P) \xrightarrow{(U.V)} (\alpha',\gamma',P') \qquad (\alpha',\gamma',Q) \xrightarrow{(U'.V')} (\alpha'',\gamma'',Q')\end{array}}{(\alpha,\gamma,P|[\{k.x|B\}]|Q) \xrightarrow{(U.V)} (\alpha'', U = U' \wedge V = V' \wedge B[U/k][V/x] \wedge \gamma', P'|[\{k.x|B\}]|Q')}\ \textsf{Par_in}$$

This rule means that a Parallel process can communicate between the left-hand-side and right-hand-side sub-processes if they can produce the same action and the action is a valid action in the middle set of the Parallel process (the Par_in rule). Other rules allow each sub-process to progress independently. For rules in this category, we replace labels by parameter pairs $(U.V)$, giving a fresh parameter pair for each transition hypothesis. This allows for separation of the constraints for each of the hypotheses. In the Par_in rule, since we need to make sure that the actions produced by the left-hand-side and right-hand-side sub-processes are equal, we also need to put into the final environment condition the condition that the two actions are equal. Also, we must add a requirement that the label (in its two parts, U and V) satisfies the set constraint of the Parallel operator. After we finish constructing the new condition, we need to verify it and see whether or not there is an assignment function ρ that models the new condition. The translation of the Parallel rules displays the main difference between the CSP and HCSP semantics. In CSP, the Parallel rules specify individual transitions that are allowed, while in HCSP, whole families of transitions are specified, hence the "Holistic" in HCSP.

In the CSP semantics, there is a certain ambiguity in dealing with ill-formed expressions. The boolean guard in the If-then-else operator is an example of this. Therefore, it needs special treatment when merging the boolean guard into the existing condition. We introduce a condition that includes the Wf operator applied to the boolean guard b of the If-then-else process to resolve the ambiguity. This means that the If-then-else process can transition if and only if the boolean guard is fully evaluated and is actually a boolean. In a given environment, the expression for the boolean guard might or might not be capable of evaluating to a boolean value, and it is important to distinguish the case of non-evaluation from either evaluating to true (enabling transition to the then process) or false (enabling transition to the else process). For example, in the HCSP process if $x < 1$ then P else Q, the else branch should be taken when $x \geq 1$, not simply when $x < 1$ does not hold, since the latter includes the case when x is a list or something other than a number. We do not want to transition to Q in the case where x is, for example, a string. However, the semantics of the parallel operator of HCSP instead uses the "satisfies" versus "fails to satisfy" meaning of boolean expressions. For instance, in the process $P|[\{k.x|k = c \wedge x < 1\}]|Q$, if the process P commits a string action of c."*hi*", the whole process can actually make a further step by a single non-communicating move in P by the original CSP transition semantics.

The third category includes all replicated operators, which in CSP are considered to be macros based on other rules. The ones we have included are the Replicated External Choice, Replicated Internal Choice and External Set Prefix operators. For example, the Replicated Internal Choice operator has the following macro definition in CSP: $\sqcap_{x:S} P = P[a_1/x] \sqcap P[a_2/x] \sqcap ...$, where $a_1, a_2, ... \in S$. We will translate the rules of these operators from CSP into HCSP by their semantic meanings. This means that we will create new rules that are semantically equivalent to the macros for these operators. For example, we create new rules for Replicated External Choice for the original CSP as follows:

$$\frac{a \in S \quad P[a/x] \xrightarrow{\tau} P'}{\Box x : S @ P \xrightarrow{\tau} (\Box x : (S - \{a\}) @ P) \Box P'} \text{Rep_ext_tau} \qquad \frac{l \neq \tau \quad a \in S \quad P[a/x] \xrightarrow{l} P'}{\Box x : S @ P \xrightarrow{l} P'} \text{Rep_ext_nor}$$

Relying upon the associativity and commutativity of Replicated External Choice in the original CSP, these rules can be seen as special instances of the rules for the corresponding binary operators. Having added these rules to CSP, when translating these rules into HCSP rules, we follow the same procedure as for the rules in the second category except that we add new hypotheses to indicate that the parameters, such as U and V in the Ext_prefix rule, are not in the existing parameter set α.

4 Correctness of HCSP with Respect to CSP

In this section, two main theorems are proved to show that the HCSP semantics is sound and complete with respect to the original CSP semantics. Our reference semantics is the CSP transition semantics introduced by Roscoe, Brookes and Walker [10], with the updated syntax of CSP-Prover [6]. All of the work described here has been formally carried out in the interactive theorem prover Isabelle/HOL [9]. Our Isabelle code may be found at `http://www.cs.illinois.edu/~egunter/fms/HCSP/hcsp.tar.gz`. The proofs of soundness and completeness of the HCSP semantics are parameterized by user-defined acts, which correspond to user-defined values in the original CSP semantics. These proofs are also parameterized by a user-defined set of process names, together with their associated processes given by *procEnv*, and a notion of *freeParams* satisfying

$$\mathit{freeParams}(\mathit{procEnv}(p)) \subseteq \mathit{freeParams}(p))$$

for every process name p.

In the Isabelle code for HCSP, the values and acts are two different types, but in this paper, by common abuse of notation, we will assume that values and acts have the same type. We use m and n to refer to values below. In the Isabelle code, we must also have a separate function for each type of construct we need to translate from HCSP to CSP. By and large, these functions are just the obvious translations. Here we will abuse notation and uniformly refer to them all as *sem*.

Before stating the soundness and completeness theorems, a very important observation is that transitions properly track the parameters introduced:

Theorem 4.1 (Well-tracked Parameters)**.** *Assume we have a transition* $(\alpha, \gamma, P) \xrightarrow{l} (\alpha', \gamma', P')$ *in HSCP such that* $\mathit{freeParams}(P) \cup \mathit{freeParams}(\gamma) \subseteq \alpha$. *Then* $\mathit{freeParams}(P') \cup \mathit{freeParams}(\gamma') \subseteq \alpha'$.

Proof. (*Sketch*) By induction on the HCSP semantics rules, since the resultant α' always contains both α and any new parameters. The replicated operators use that $\mathit{freeParams}(P) \subseteq \alpha$ implies $\mathit{freeParams}(P[U/x]) \subseteq \alpha \cup \{U\}$. □

The proofs of soundness and completeness require the following facts describing how parameters, assignments, substitution, and translation interact:

Lemma 4.2. *If $\rho(U) = n$, then $sem(\rho, P[U/x]) = sem(\rho, P[n/x])$.*

Lemma 4.3. *If $\rho \models B[U/x][V/y]$ and $sem(\rho,U)$ and $sem(\rho,V)$ are well defined, then $(\rho(U).\rho(V))$ is in the set $sem(\rho, \{x.y|B\})$.*

Lemma 4.4. *If $freeParams(P) \subseteq \alpha$, $\rho'|_\alpha = \rho|_\alpha$, then $sem(\rho,P) = sem(\rho',P)$.*

With these, we can show that every interpretation of every transition we can take in HCSP is valid in CSP.

Theorem 4.5 (Soundness)**.** *For all HCSP processes P, P′, assignments ρ, environment conditions γ, γ' such that $\rho \models \gamma$ and $\rho \models \gamma'$, and parameter set α such that $freeParams(P) \cup freeParams(\gamma) \subseteq \alpha$, if $(\alpha,\gamma,P) \xrightarrow{l} (\alpha',\gamma',P')$, then $sem(\rho,P) \overset{sem(\rho,l)}{\longrightarrow} sem(\rho,P'))$.*

Proof. (*Sketch*) By induction on the HCSP semantics rules. It is worth noting that each assignment function ρ is a total function, and thus gives some value to every parameter, regardless of whether the parameter has been included in the parameter set α.

Each rule in HCSP has a corresponding rule in CSP. When we prove soundness, we must show that for each rule in HCSP, if the hypotheses of the rule are valid in HCSP, then the corresponding hypotheses of the corresponding CSP rule are also valid. This follows from Theorem 4.1 and the fact that the environment condition γ only becomes logically stronger with each transition. Handling the rules for the replicated operators requires use of Lemma 4.2. The hypotheses in the CSP rules include side conditions that have become incorporated in the environment condition in the derived HCSP rule. It is therefore necessary to prove these side conditions from the assumption of the validity of the initial environment condition, and the specifics of the transition in HCSP. For this, we need to make use of Lemma 4.3. □

As yet, HCPS does not support all processes in CSP. In particular, there is no support for the Renaming operator. As a result, our completeness theorem is restricted to that portion of CSP supported by HCSP.

Theorem 4.6 (Relative Completeness)**.** *Let P be an HCSP process, ρ be an assignment, γ be an environment condition such that $\rho \models \gamma$, and α be a parameter set such that $freeParams(P) \cup freeParams(\gamma) \subseteq \alpha$. Further suppose we have a CSP process T and a value i such that $sem(\rho,P) \xrightarrow{i} T$ in CSP semantics. Then there exist an HCSP process P′, an assignment ρ', an environment condition γ', a parameter set α', and a label l such that $i = sem(\rho',l)$, $\rho'|_\alpha = \rho|_\alpha$, $T = sem(\rho',P')$, $\rho' \models \gamma'$ and $(\alpha,\gamma,P) \xrightarrow{l} (\alpha',\gamma',P')$.*

Proof. (Sketch) By induction on the CSP semantics rules. For each rule in HCSP, the first thing we need to do is to use Theorem 4.1 and Lemma 4.4 to prove the theorem hypotheses for the inductive instances. Then we will interpret l, P' α', γ' and ρ' according to the corresponding HCSP rules and show that the interpretation is correct. For each rule, since the only parameters generated from the evaluation of an HCSP process are parameters not in set α, we can always make a new assignment function ρ' based on the current ρ and $\rho'|_\alpha = \rho|_\alpha$. To show that $i = sem(\rho',l)$, we need to do a case analysis on all possible labels an HCSP process can produce given the fact that there is always an interpretation for a given free parameter U. Finally, in some rules, we need to show that the ρ' we are selecting can properly model the post-condition of the environment. To prove this, we must use the same technique as we use in reasoning about the post-condition in the Soundness theorem and a uniform group of lemmas represented by Lemmas 4.3. □

5 Simulator

To put the theory of HCSP into practice, we have implemented an HCSP simulator with a rich mutually recursive datatype for actions and propositions in Isabelle. We also extract the Ocaml code from Isabelle directly. The core of the simulator is included with the package for the soundness and completeness theorems. In the simulator, we have limited the propositions to qualifier-free first order logic with Presburger arithmetic in order to maintain decidability. In doing so, we render the single-step transition relation computable as a function generating a finite set of possibilities. We then represent all possible traces with a lazy stream data structure supporting back-tracking. This enables us to incrementally compute the requirements for a given trace, which can be inspected at each step, and can be back-tracked when the requirements are proven to be unsatisfiable. Using the HCSP semantics, we can indefinitely delay the calculation of a specific trace using trace patterns and pre- and post-conditions, until one trace pattern / condition is selected. At this point, satisfiability analysis can be used to generate an instance trace, if such is desired.

In the case of the medical mediator, we were able to use the simulator (semantically embedded in Isabelle) to enumerate the possible trace patterns for the System, and to verify that all traces satisfying each pattern-condition so enumerated satisfy a pattern-condition of the Saftey process.

6 One More Example

$$\mathsf{Clicker}(c,r) = \bigsqcap_{\substack{s:s>0\wedge\\ s\leq N}} K.r.c.s \to \mathsf{Clicker}(c,r) \qquad \mathsf{Broadcast}(r) = \bigsqcup_{c:true} K.r.c?s : s>0\wedge s\leq N \to K.r.s \to \mathsf{Broadcast}(r)$$

$$\mathsf{Room}(r) = (\mathsf{Clicker}(c1_r,r)|[\{\}]|\mathsf{Clicker}(c2_r,r))|[\{k.x|\exists c\ s.\ k = K.r.c\wedge x = s\}]|\mathsf{Broadcast}(r) \qquad \mathsf{Center} = \bigtriangleup_{\substack{r:r>0\wedge\\ r\leq 25}} \mathsf{Room}(r)$$

Figure 7: Example

Besides the small example in Fig. 1 and the medical mediator example from Gunter *et al.* [3], there are many other real implementations that can benefit from modeling in the HCSP system. Generally speaking, every real model with several users trying to access one or more copies of a very large database can benefit from the HCSP system. A song broadcasting system is one such example. Song broadcasting systems are used in entertainment businesses such as discos and karaokes to allow people to select songs from a large database. Such systems typically have a large collection of songs; a collection in excess of 100,000 would not be uncommon. A typical karaoke bar has about 25 rooms for separate entertainment. Typically, each room has two remote clickers for selecting the next song to be played. After a user selects a song, the remote clicker will send the song selection to the song broadcasting system and the song broadcasting system will play the chosen song in the room. Since only one song will be broadcast at a time, if two people send their selected songs through clickers at the same time, only the signal from one of the clickers will honored immediately, while the other signal will be delayed for later action.

We model the karaoke center in CSP in the Figure 7. We will use the Clicker process to model the behavior of a remote clicker in a karaoke room. The input argument c is the clicker's identity and r represents the room to which the clicker belongs. The Clicker allows the user to select any song in the database, represented by the second Replicated Internal Choice operator over $s : s > 0 \wedge s \leq N$, where N is the size of the song database.

After the song has been decided, the Clicker process sends out the message $K.r.c.s$ by the Action Prefix operator, where K is a channel representing the karaoke center. The whole sequence $K.r.c.s$ means that in room r of the karaoke center clicker c was used to select the song s. We will use the Broadcast process to represent the behavior of the song broadcasting system in a room. It will wait for the signal from one of the clickers telling which song to play. These choices are represented by the two Replicated External Choice operators. Having received the song request, it will broadcast the song in the room originating the request. This is represented by Action Prefix operator with the sequence $K.r.s$. The Room process with id r represents the behavior of the room as a whole. The room is a parallel composition of the two clickers in the room (with no communication between them) and the broadcast system, synchronizing on the music requests. Finally, the Center process models the whole karaoke center. In the Center process, we use a Replicated Interleaving operator to represent the behavior in all twenty-five rooms. It indicates that the twenty-five rooms have no communication interactions.

The capital letter N refers to an arbitrary number to represent the size of the database that contains all songs in the song broadcasting system. Typically, we know that the number N is a large number, but we do not know exactly how large it is. In order to verify properties in the system, such as safety and deadlock-freedom, it is better to leave the number N to be unspecified. Obviously, original CSP-based tools, such as FDR2 and ProBE cannot deal with the case when N is unspecified. Even specified, but very large values can not be handled. For example, when we test the case when N is 100,000 in ProBE, the program fails to start. On the other hand, the HCSP semantics can handle the model with only a few transitions. This fact indicates that the HCSP semantics can be useful in cases previously not handled and it will expand the category of CSP programs for which we can perform automated analysis.

7 Related Work

Currently, there are several existing CSP simulators and model checkers based on the original CSP transition semantics. CSPM [11] gives a standard CSP syntax and semantics in machine readable form, introduced by Bryan Scattergood, which is based on the transition semantics introduced by Brookes and Roscoe [10]. It provides a standard for many CSP tools, including FDR2, the industry standard for CSP model checkers [8]. FDR2 uses the traditional view of actions as single elements, and so tends to generate a large number of states when comparing multiple possibilities for actions. ProBE [7] is a simulator created by the same group, which simulates a CSP process by listing all the actions and states one by one as a tree structure [7]. Additionally, traditional CSPM-semantics based tools such as ProBE and FDR2 treat some operators, especially replicated operators, as macros, and hence, it is impossible for these tools to generate traces when the replicated set is infinite.

The medical mediator project by Gunter *et al.* [3] provides a example of the advantages of HCSP over CSP-semantics based tools. The main goal of the medical mediator project is to prove that the set of traces of the process $System|[Vis]|Given$ is a subset of the traces of $Safety|[Vis]|Given$, where Vis is a set defined as $\{y.(\exists n\ d\ m\ x.\ y = RFIDChan_d^{n,m}.x) \vee (\exists m\ z.\ y = EHRBECh_m.z)\}$. (See Fig. 3.) This requires exploring all possible traces generated by the System process. Tools based on the original CSP semantics, such as FDR2, fail when dealing with large or unbounded sets. For example, the Med process as given does not put any restrictions on the sets of values that may be received over various channels. The simulator we have built based on HCSP semantics benefits from being able to handle such large or unbounded sets uniformly as single actions, thus avoiding state explosion problems. In addition to the simulator, the work in this paper was all developed in the interactive theorem prover Isabelle. This means that we have the additional recourse of using interactive proof methods such as structural and rule

induction when analyzing the traces of a system such as the medical mediator. While FDR2 can only check the above safety property for the combination of two patients, two nurses, two devices and two medical mediators, using induction with HCSP semantics in Isabelle, we can prove the safety property of the same process for an arbitrary number of patients, nurses, devices, and medical mediators.

CSP-Prover is a theorem proving tool built on top of Isabelle [6]. It provides a denotational semantics of CSP in Higher-Order Logic. CSP-Prover uses functions in places of terms with bindings to avoid issues of α-conversion and substitution. This formulation means that CSP-Prover can express terms that are not expressible in standard CSP syntax. Despite its extra expressive capability, CSP-Prover can not directly handle proving the desired safety property for the medical mediator, again due to its direct implementation of the original CSP semantics. While carrying out proofs manually can take advantage of the theorem proving setting, much of the automation fails to be applicable to code such as was given in the medical mediator example.

CSPsim [1] is another simulator based on the FDR2 standard. Its major innovation is the use of "lazy evaluation". The basic idea of CSPsim is to keep track of all the current actions, then compare them with the actions of the outside world and only select the possible executable actions for the very next step [1]. The phrase "lazy evaluation" refers to a pre-processing step in which CSPsim selects some processes that contain fewer actions and generates some conditions in advance. After that, CSPsim evaluates the whole program based on these conditions. For instance, if a program has a STOP process in a branch of the parallel operator, the next possible action can only be a τ-action. Hence, the CSPsim will select the STOP process as the very next step, resulting in fast feedback. Nonetheless, after generating all the possible states for the next step, it still must compare possible actions with those provided by the outside world one at a time instead of considering a group of properties.

8 Conclusion and Future Work

In this paper we have presented HCSP, a new semantics for CSP. HSCP provides an alternative way to model CSP processes by viewing transitions as bundles of the original transitions, where all transitions in the bundle can be described by a uniform property derived from the process. By this translation, we can allow HCSP-based tools to run some CSP programs which are not able to run in the original CSP-based tools. We have shown the HCSP semantics to be equivalent to the original CSP transition semantics. We have also presented an HCSP-based simulator, which is extracted directly from the Isabelle code for the HCSP semantics, and shown that it is not only theoretically powerful, but also practical to implement. By using the medical mediator project [3] as an example, we show that the HCSP semantics implemented with Isabelle's system for inductive rule definition and reasoning can overcome some difficulties that traditional CSP-semantics-based tools cannot handle.

The ideas represented in this alternate semantics are applicable beyond the example of an alternate semantics for CSP. Roughly, the methodology applied here may be described as a two phase transformation. In the first phase, we transform a substitution-based transition relation into a corresponding environment-based transition relation. In the second phase, we transform individual transitions into symbolic transition patterns, replacing environments by predicates over system variables giving pre- and post-conditions for our transitions. These conditions are derived from the tests present in the processes being transitioned, and the side conditions of the original transition rules. This methodology has a particularly significant pay-off for CSP, where the behavior of many operators, such as the replicated choice operators, are predicated on the satisfaction of complex boolean expressions. However, we believe this methodology could lend useful insights to the transition semantics of other systems as well.

For further study, we are interested in combining the ability to reason locally about code fragments by induction with an automatic model checker. The model checker should be directly related to the HCSP semantics in Isabelle. In this way, we can prove the correctness of our HCSP model checker with respect to the HCSP semantics. The new model checker will be able to check some programs that are hard to analyze with traditional CSP tools. In another direction, we will try to build a dependent session type system for the HCSP framework. We believe there may be interesting synergies between dependent session types and symbolic execution traces.

9 Acknowledgments

This material is based upon work supported in part by NASA Contract NNA10DE79C and NSF Grant 0917218. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NASA or NSF.

References

[1] Phillip J. Brooke & Richard F. Paige (2007): *Lazy Exploration and Checking of CSP Models with CSPsim.* In Alistair A. McEwan, Steve A. Schneider, Wilson Ifill & Peter H. Welch, editors: *CPA*, *Concurrent Systems Engineering Series* 65, IOS Press, pp. 33–49. Available at `http://dblp.uni-trier.de/db/conf/wotug/cpa2007.html#BrookeP07`.

[2] Stephen D. Brookes, C. A. R. Hoare & A. W. Roscoe (1984): *A Theory of Communicating Sequential Processes*. *J. ACM* 31(3), pp. 560–599. Available at `http://doi.acm.org/10.1145/828.833`.

[3] Elsa L. Gunter, Ayesha Yasmeen, Carl A. Gunter & Anh Nguyen (2009): *Specifying and Analyzing Workflows for Automated Identification and Data Capture*. In: *HICSS*, pp. 1–11. Available at `http://dx.doi.org/10.1109/HICSS.2009.402`.

[4] C. A. R. Hoare (1969): *An axiomatic basis for computer programming*. *Commun. ACM* 12(10), pp. 576–580, doi:10.1145/363235.363259. Available at `http://doi.acm.org/10.1145/363235.363259`.

[5] C. A. R. Hoare (1985): *Communicating sequential processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[6] Yoshinao Isobe & Markus Roggenbach (2006): *A Complete Axiomatic Semantics for the CSP Stable-Failures Model*. In Christel Baier & Holger Hermanns, editors: *CONCUR 2006 Concurrency Theory*, *Lecture Notes in Computer Science* 4137, Springer Berlin / Heidelberg, pp. 158–172. Available at `http://dx.doi.org/10.1007/11817949_11`. 10.1007/11817949_11.

[7] Formal Systems (Europe) Ltd (2003): *Process Behaviour Explorer. ProBE User Manual*. In: *ProBE User Manual*.

[8] Formal Systems (Europe) Ltd. (2010): *Failures-Divergence Refinement. FDR2 User Manual*. In: *FDR2 User Manual*.

[9] Lawrence C. Paulson (1990): *Isabelle: The Next 700 Theorem Provers*. In P. Odifreddi, editor: *Logic and Computer Science*, Academic Press, pp. 361–386.

[10] A. W. Roscoe, S. D. Brookes & D. J. Walker (1986): *An Operational Semantics for CSP*. Technical Report, Oxford University Computing Laboratory. Available at `http://www.cs.ox.ac.uk/people/bill.roscoe/publications/26.ps`.

[11] J.B. Scattergood (1998): *The Semantics and Implementation of Machine-Readable CSP*. D.phil., Oxford University Computing Laboratory.